




Chapitre 3

Modèles de Cycle de vie logiciel et méthodes de développement



Objectifs du Chapitre 3

Partie 1



- Définir les étapes d'un cycle de vie logiciel.
- Expliquer différents modèles de cycles de vie logiciel.
- Choisir le cycle de vie adapté pour un type de projet donné.



Plan chapitre 3

Partie 1



I. Introduction

II. Définition du Cycle de vie logiciel

III. Modèles de cycle de vie logiciel :

- ✓ Modèle de cycle de vie en cascade
- ✓ Modèle de cycle de vie en V
- ✓ Modèle de cycle de vie par prototypage
- ✓ Modèle de cycle de vie incrémental
- ✓ Modèle de cycle de vie en spirale

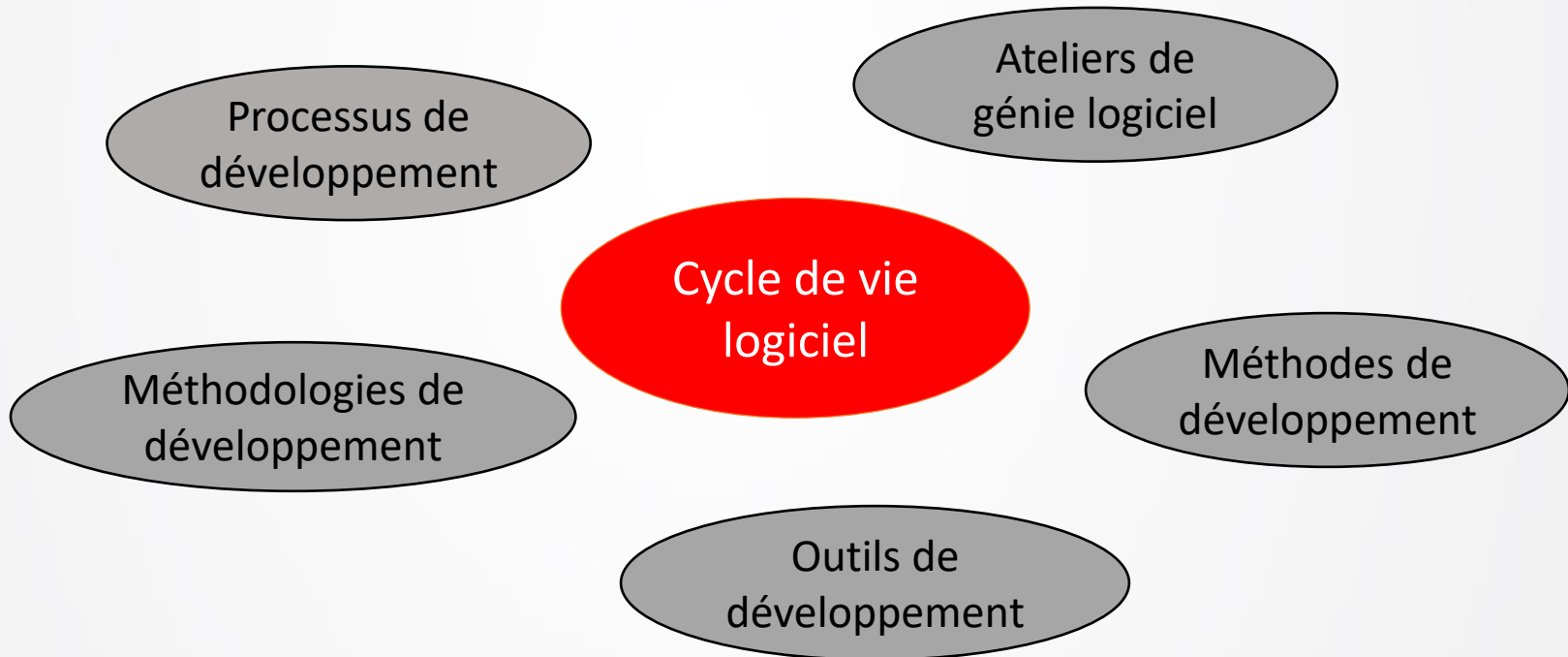


INTRODUCTION



Question : Que va-t-on faire pour construire un logiciel de qualité?

Réponse : Adopter des bonnes pratiques





Définition : Cycle de vie logiciel



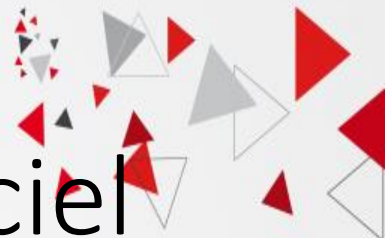
Définition du cycle de vie

Ensemble d'étapes qui composent le processus de développement et d'utilisation du logiciel





Modèle de Cycle de vie logiciel



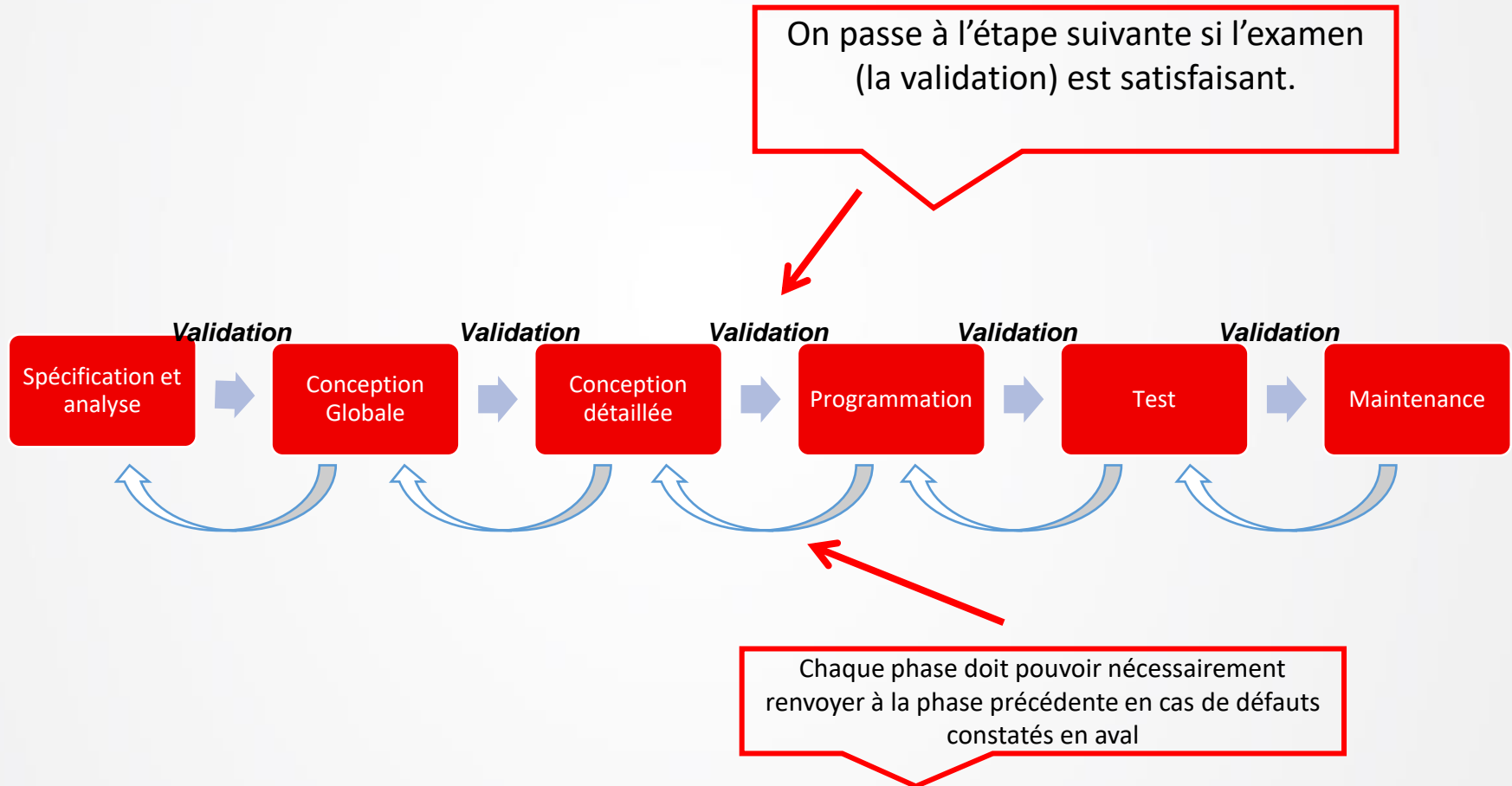
Modèle de cycle de vie :

- Modélisation conventionnelle de la succession d'étapes qui préside à la mise en œuvre d'un produit logiciel.
- Plusieurs modèles : en cascade, en V, en spirale, par prototypage, incrémental, etc.

Les objectifs de ces modèles :

- Représenter le processus de développement de manière graphique et physique.
- Donner une structure autour de laquelle les activités d'assurance qualité peuvent être construites.

Modèle de cycle de vie en cascade (1/3)





Modèle de cycle de vie en Cascade (2/3)



- **Principe :**

- Le développement se divise en étapes.
- Chaque étape se termine à une **date précise**.
- Des documents/programmes sont produits à **la fin de chaque étape**.
- Le résultat de chaque étape est soumis à un examen (Validation).
- On passe à l'étape suivante **si l'examen est satisfaisant**.
- Chaque phase doit pouvoir nécessairement renvoyer à la phase précédente en cas de défauts constatés en aval (par exemple, en cas d'erreur découverte lors des tests, il est nécessaire de retourner à la phase de programmation)



Modèle de cycle de vie en cascade (3/3)



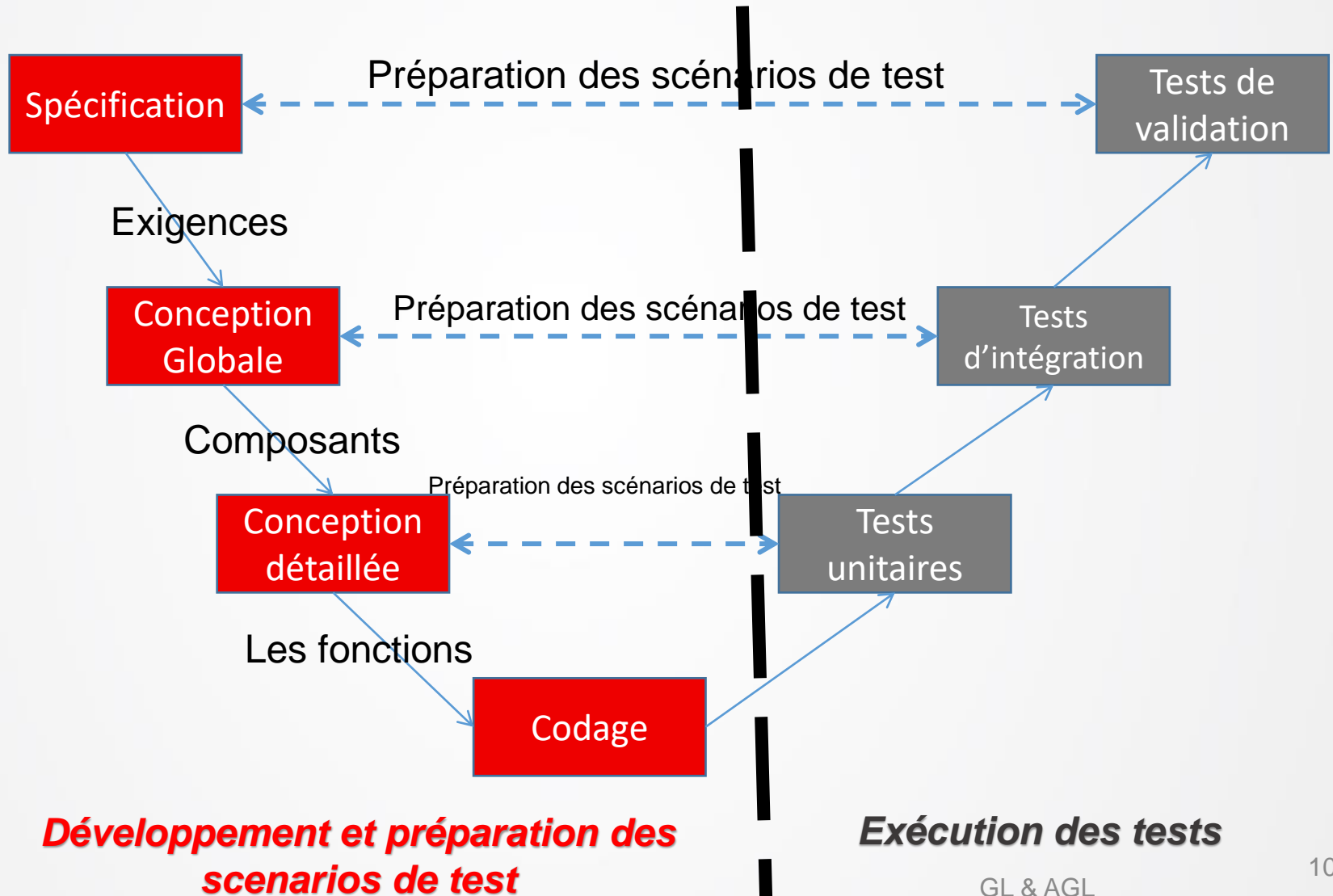
- **Avantage :**

- ✓ Simple à mettre en place.
- ✓ Permet de garantir l'existence d'une documentation bien structurée

- **Inconvénients :**

- ✓ Détection tardive des erreurs => Augmentation du coût de la correction, etc.
- ✓ N'est pas efficace pour les projets qui exigent la prise en considération de risques.

► Modèle de cycle de vie en V (1/3)





► Modèle de cycle de vie en V (2/3)

- **Principe :**

- ✓ Les premières étapes préparent les dernières.

- **Interprétations :**

- ✓ 2 sortes de dépendances entre les étapes du modèle de cycle de vie en V .
 - ✓ Enchaînement séquentiel (modèle en cascade) de gauche à droite, les résultats des étapes de départ sont utilisés par les étapes d'arrivée.

► Modèle de cycle de vie en V (3/3)

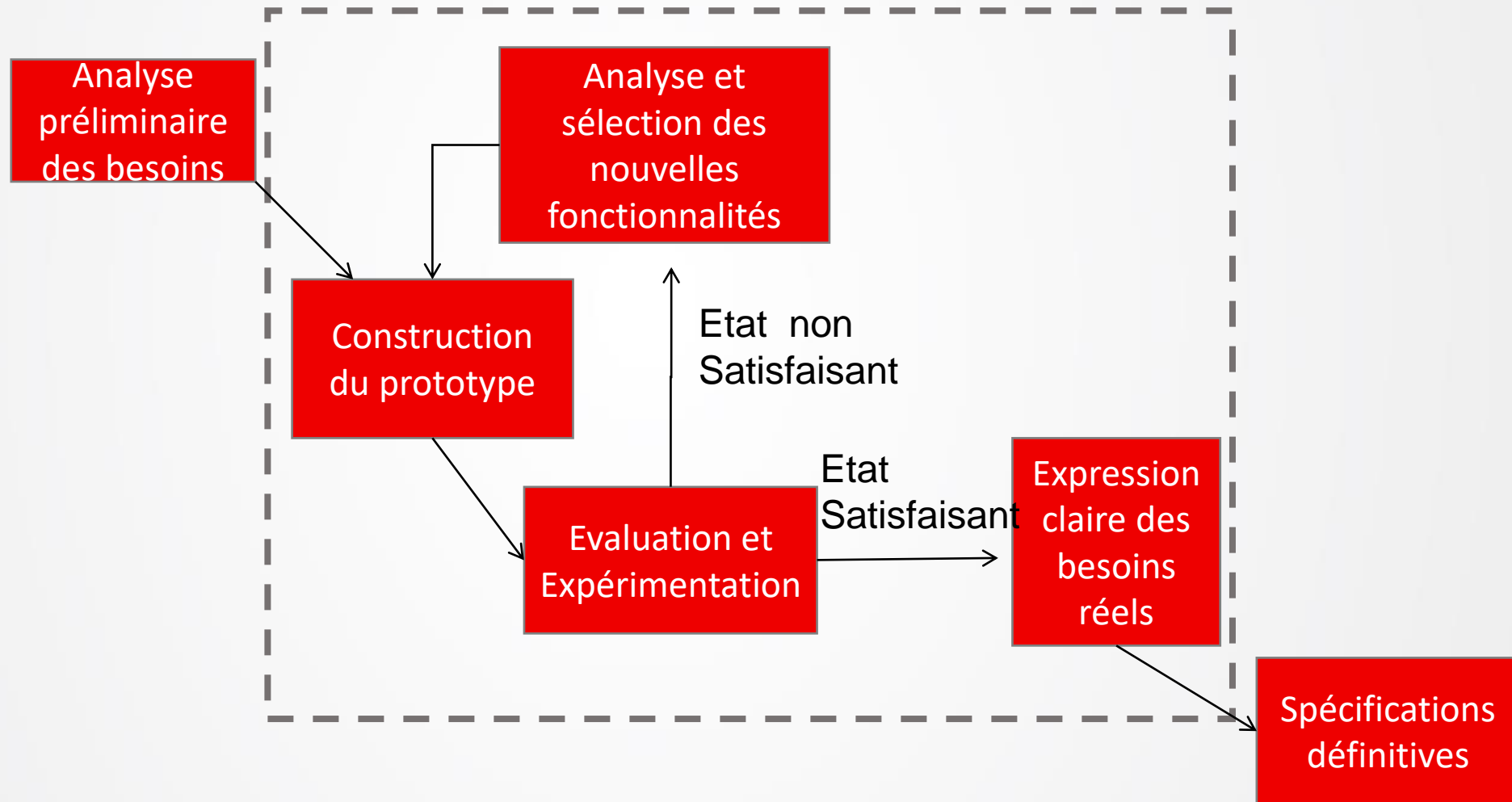
- **Avantages**

- ✓ Vérification objective des spécifications.
- ✓ Modèle plus réaliste que le modèle en cascade.
- ✓ Epruvé pour des grands projets (le modèle le plus utilisé).

- **Inconvénients:**

- ✓ N'est pas efficace lorsque les besoins du client ne sont pas stables.
- ✓ N'est pas efficace pour les projets qui exigent la prise en considération de risques.

Modèle de cycle de vie par prototypage (1/3)



► Modèle de cycle de vie par prototypage (2/3)



- **Principe :**

- ✓ Les spécifications initialement données par le client sont d'ordre général et ne donnent que des grandes lignes.
- ✓ Raffinement des spécifications, des fonctionnalités et performances par des prototypes successifs.
- ✓ Les prototypes servent comme catalyseur pour mieux cerner les estimations et les coûts de développement.
- ✓ Invitation et implication du client à intervenir dans l'expression de son besoin en fonction de l'évolution du prototype.
- ✓ L'analyse de chaque prototype conduit à un développement souvent rapide en but de converger rapidement.
- ✓ Expérimentation de plusieurs techniques de réalisation.



Modèle de cycle de vie par prototypage (3/3)



- **Avantages :**

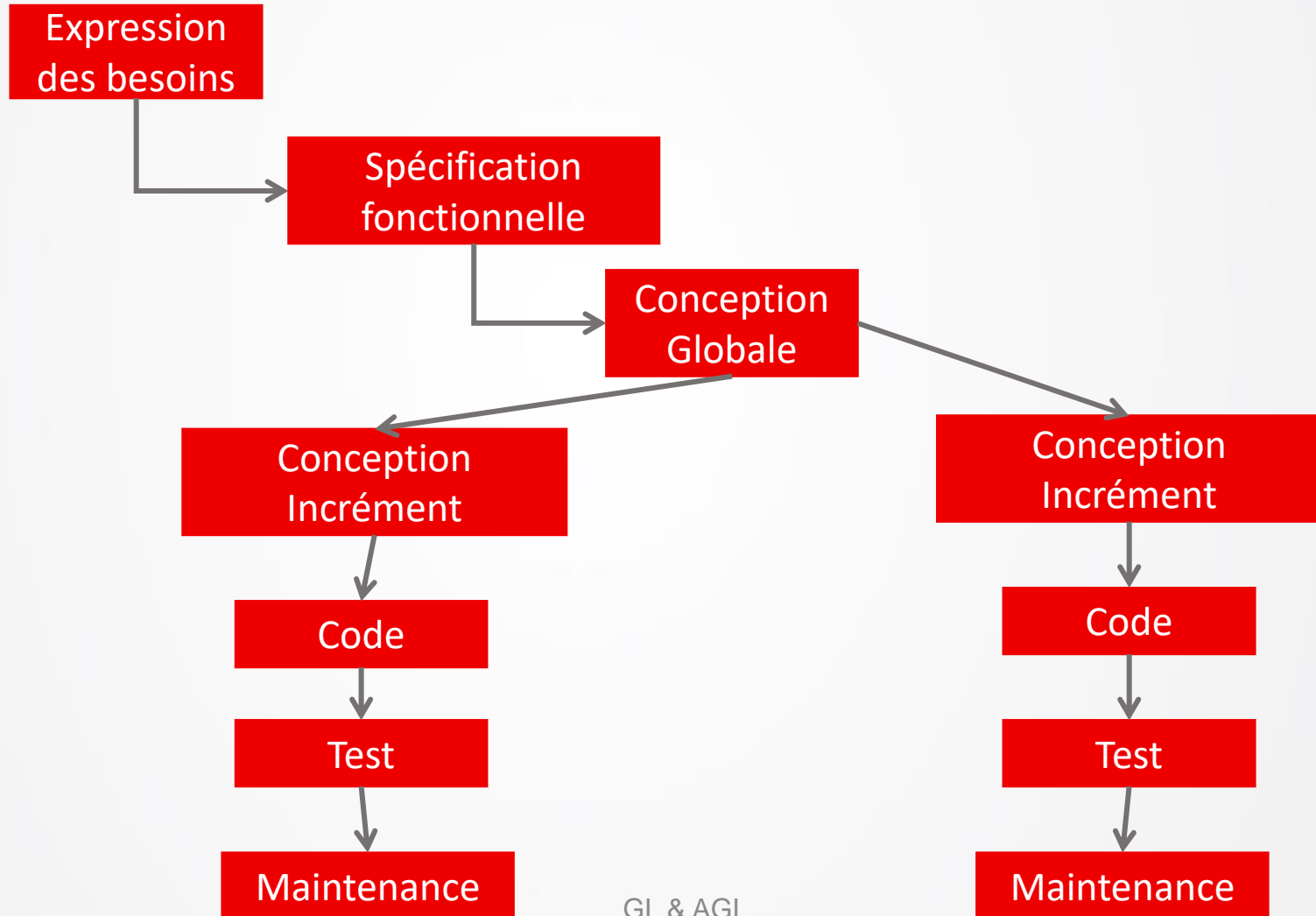
- ✓ Le client participe activement dans le développement du produit
- ✓ Le client reçoit des résultats tangibles rapidement (expérimentation rapide des fonctionnalités souhaitées par les utilisateurs).
- ✓ Introduction d'un feedback immédiat de la part des utilisateurs.
- ✓ Amélioration de la communication entre l'équipe de développement et le client.
- ✓ Permet d'éviter l'effet Tunnel.

- **Inconvénients :**

- ✓ Le coût du projet peut rapidement s'écarter.
- ✓ Il est impossible d'estimer les délais au démarrage du projet.



Modèle de cycle de vie incrémental (1/3)





Modèle de cycle de vie incrémental (2/3)



- **Principe :**

- ✓ Développer des applications en étendant PROGRESSIVEMENT ses fonctionnalités.
- ✓ La stratégie consiste à développer le logiciel par extension successive à partir d'un produit « Noyau » du logiciel.
- ✓ Permet d'éviter de TOUT CONCEVOIR, de TOUT TESTER comme l'approche en cascade.
- ✓ Cette technique a des répercussions sur la répartition des efforts en fonction du temps.



Modèle de cycle de vie incrémental (3/3)



- **Avantages :**

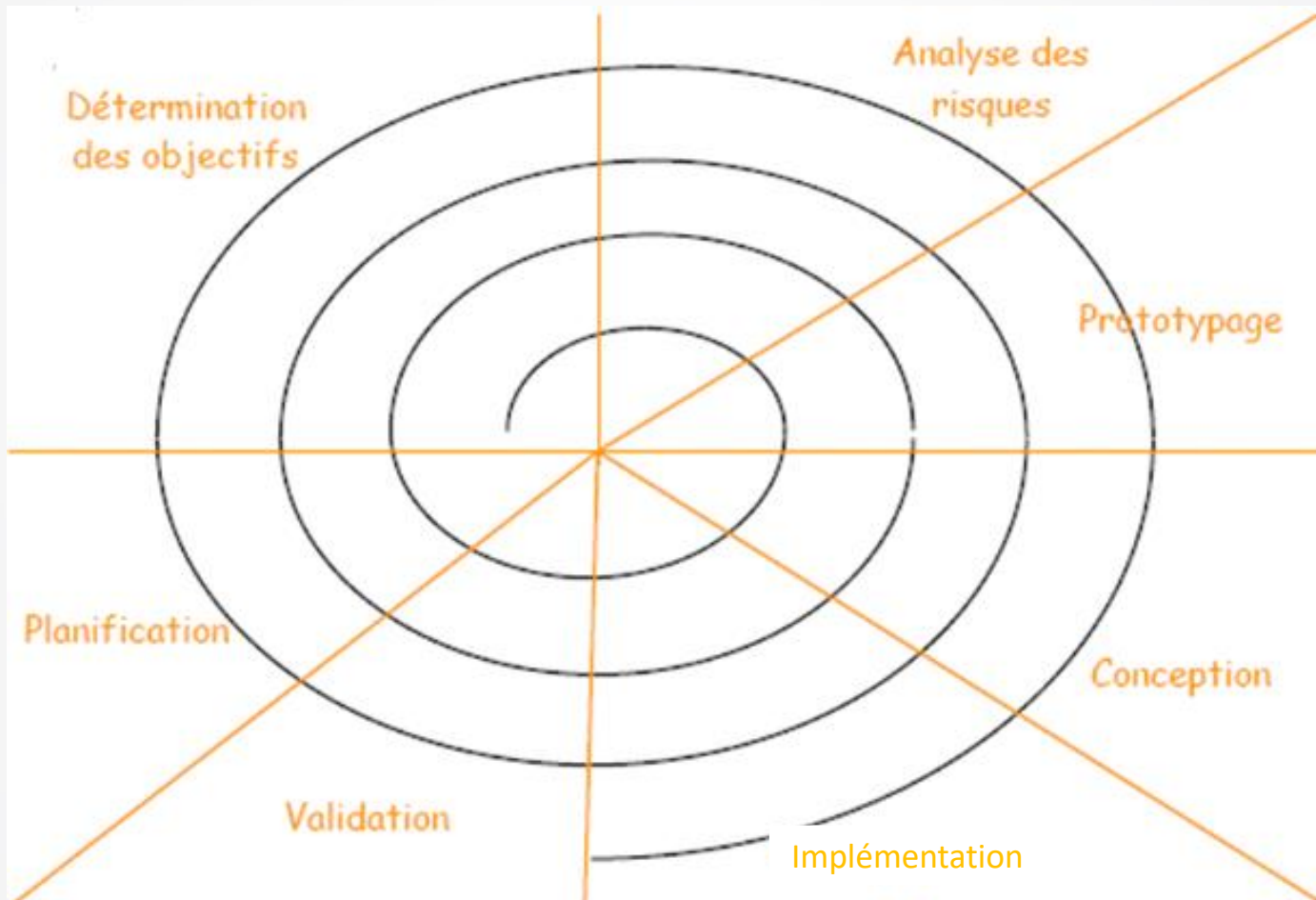
- ✓ Le développement du projet sous forme d'incrément est plus simple.
- ✓ Les intégrations sont progressives.
- ✓ Il peut y avoir des livraisons et des mises en service après chaque intégration d'incrément.
- ✓ Permet d'optimiser le temps et le partage des tâches.
- ✓ Diminution d'effort pendant la phase de test d'intégration.

- **Inconvénients :**

- ✓ Les incréments doivent être indépendants aussi bien fonctionnellement qu'au niveau des calendriers de développement.
- ✓ La difficulté de fixation des incréments dès le début du projet.



Modèle de cycle de vie en Spirale (1/3)





Modèle de cycle de vie en Spirale (2/3)



- **Principe :**
 - ✓ développement itératif (prototypes).
- **Interprétation :** Chaque mini-cycle se déroule en 4 phases
 1. Analyse des besoins, Spécification.
 2. Analyse des risques, Alternatives, Maquettage.
 3. Conception et Implémentation de la solution retenue.
 4. Vérification, Validation, Planification du cycle suivant.



Modèle de cycle de vie en Spirale (3/3)



- **Avantages :**

- ✓ Nouveau : analyse des risques, maquettes, prototypages
- ✓ Modèle complet, complexe et général.
- ✓ Effort important pour la mise en œuvre.
- ✓ Utilisé pour des projets innovants ou à risques.

- **Inconvénients :**

- ✓ Difficile à appliquer.
- ✓ Beaucoup de temps.
- ✓ Coût élevé.

Objectifs du Chapitre 3

Partie 2



- Distinguer entre une méthode lourde et une méthode agile.
- Illustrer les principales caractéristiques d'une méthode de développement – lourde ou agile.
- Explorer les instances du processus unifié 2TUP et RUP.
- Organiser et planifier un projet informatique conformément à une méthode de développement donnée.



Plan chapitre 3

Partie 2



I. Méthodes lourdes

- ✓ Le processus unifié.
- ✓ RUP (Rational Unified Process)
- ✓ 2TUP (Two Track Unified Process)

II. Méthode agile

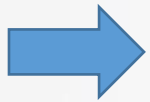
- ✓ Scrum



Méthodes lourdes



- **Le processus unifié (UP) est une méthode générique de développement de logiciels.**



Cette méthode nécessite donc d'être adaptée à chacun des projets pour lesquels elle sera employée.

- **Caractéristiques de UP :**
 - Piloté par les cas d'utilisation.
 - Itératif et incrémental.
 - Centré sur l'architecture.
 - Orienté risques.



Méthodes lourdes

- ***UP est piloté par les cas d'utilisation :***
 - Système analysé, conçu et développé pour des utilisateurs.
 - Tout doit donc être fait en adoptant le point de vue utilisateur.
- ***UP est centré sur l'architecture :***
 - L'architecture du système est décrite à l'aide de différentes vues.
 - L'architecte procède de manière incrémentale :
 - Il commence par définir une architecture simplifiée qui répond aux besoins classés comme prioritaires
 - Puis définit à partir de l'architecture simplifiée les sous-systèmes de manière beaucoup plus précise.



Méthodes lourdes

- ***UP est itératif et incrémental :***
 - Le travail itératif permet à l'équipe de capitaliser à chaque cycle les enseignements du cycle précédent.
 - En procédant de manière itérative, il est possible de découvrir les erreurs et les incompréhensions plus tôt.
 - Le feedback de l'utilisateur est aussi encouragé et les tests effectués à chaque utilisation permettent d'avoir une vision plus objective de l'avancement du projet.
- ***UP est orienté risques :***
 - Identifier les risques.
 - Maintenir une liste de risques tout au long du projet.



Méthodes lourdes

- **Composantes de UP :**

- **4 phases :**

- Etude d'opportunité/ Inception/ Pré-etude..
 - Elaboration.
 - Construction.
 - Transition.

- **Ensemble d'activités:**

- Expression des besoins.
 - Analyse.
 - Conception.
 - Implémentation.
 - Test.
 - Déploiement

- **Ensemble d'itérations :** circuit de développement aboutissant à un livrable



Le Processus Unifié (UP)

- **Les principales activités dans UP :**
 - **Expression des besoins :**
 - Identification des besoins fonctionnels.
 - Identification des besoins non fonctionnels.
 - **Analyse :**
 - Formalisation du système à partir des besoins.
 - Modélisations de diagrammes statiques et dynamiques.
 - Vue logique du système.
 - **Conception :**
 - Définition de l'architecture du système.
 - Etendre les diagrammes d'analyse.
 - Prise en compte des contraintes de l'architecture technique.



Le Processus Unifié (UP)

- **Implémentation :**

- Production du logiciel :
 - Composants.
 - Bibliothèques.
 - Fichiers.
 - Etc.

- **Test :**

- Vérifier l'implémentation de tous les besoins (fonctionnels et non fonctionnels).
- Vérifier l'interaction entre les objets.
- Vérifier l'intégration de tous les composants.
- Différents niveaux de tests (unitaires, d'intégration, de performance, etc.).



Le Processus Unifié (UP)

- **Les phases de UP :**

- 1. Etude d'opportunité/ Inception/ Pré-Etude :**

- Cette phase pose la question de la faisabilité du projet, des frontières du système, des risques majeurs qui pourraient mettre en péril le projet.
- A la fin de cette phase, est établi un document donnant une vision globale des principales exigences, des fonctionnalités clés et des contraintes majeures.
 - Environ 10 % des cas d'utilisation sont connus à l'issue de cette phase.
- Il convient aussi d'établir une estimation initiale des risques, un "Project Plan", un "Business Model".



Le Processus Unifié (UP)

2. Elaboration :

- Reprise des résultats de la phase d'incubation.
- Spécification détaillée des cas d'utilisation.
- Détermination de l'architecture de référence.

3. Construction :

- Construction d'une première version du produit (version bêta ainsi qu'une version du manuel utilisateur).
- Construction de tous les cas d'utilisation.

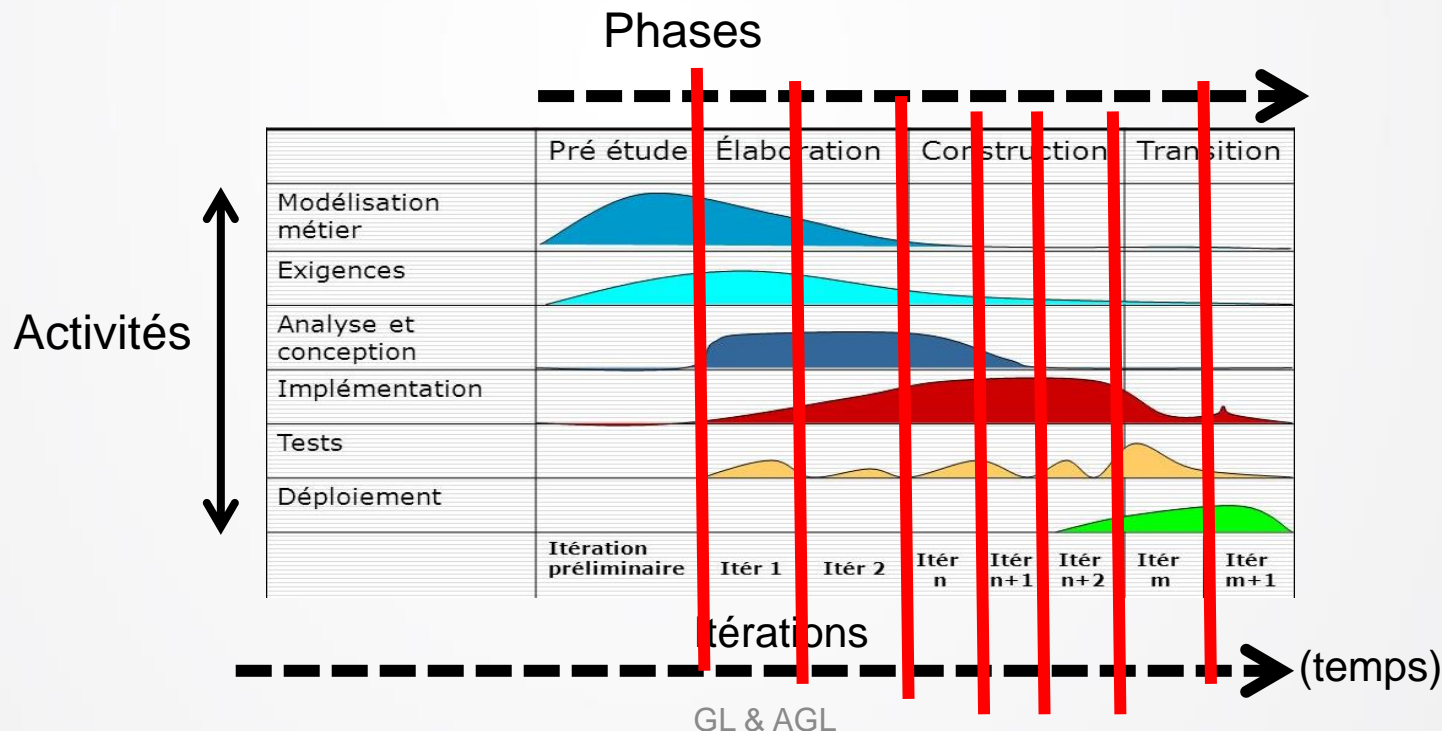
4. Transition :

- Test et correction des anomalies.
- Préparer la version finale du produit.
- Déploiement du produit.

► RUP (Rational Unified Process)



- RUP est une instance / implémentation de UP.
- RUP est une démarche de développement qui est souvent utilisé conjointement au langage UML.
- RUP implémente les caractéristiques de UP.

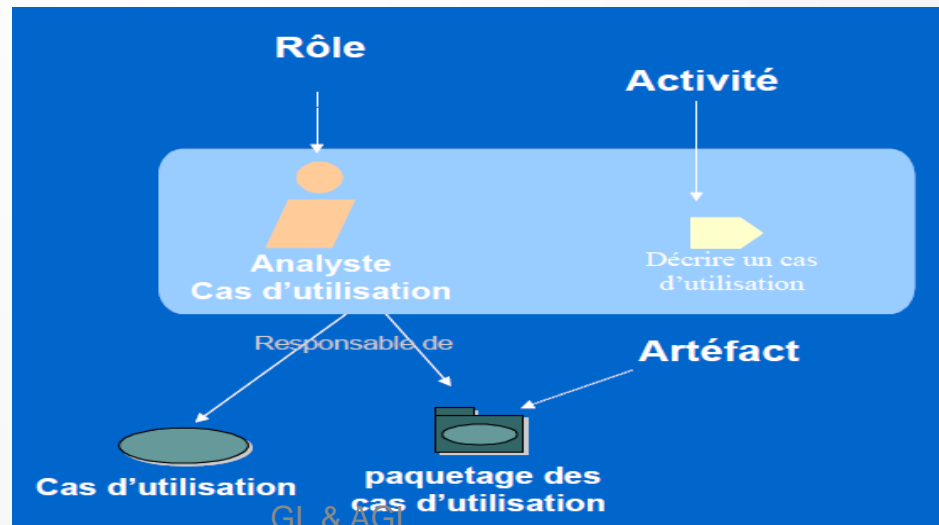


► RUP (Rational Unified Process)





- RUP: Définition et notation

- **Artéfact** : Élément d'information, produit ou utilisé lors d'une activité de développement logiciel (modèle, source, etc.)
- **Activité** : Opération exécutée au sein d'un état. Une activité peut être interrompue.
- **Rôle** : Comportement et responsabilités d'un ensemble de personnes.



► RUP (Rational Unified Process)



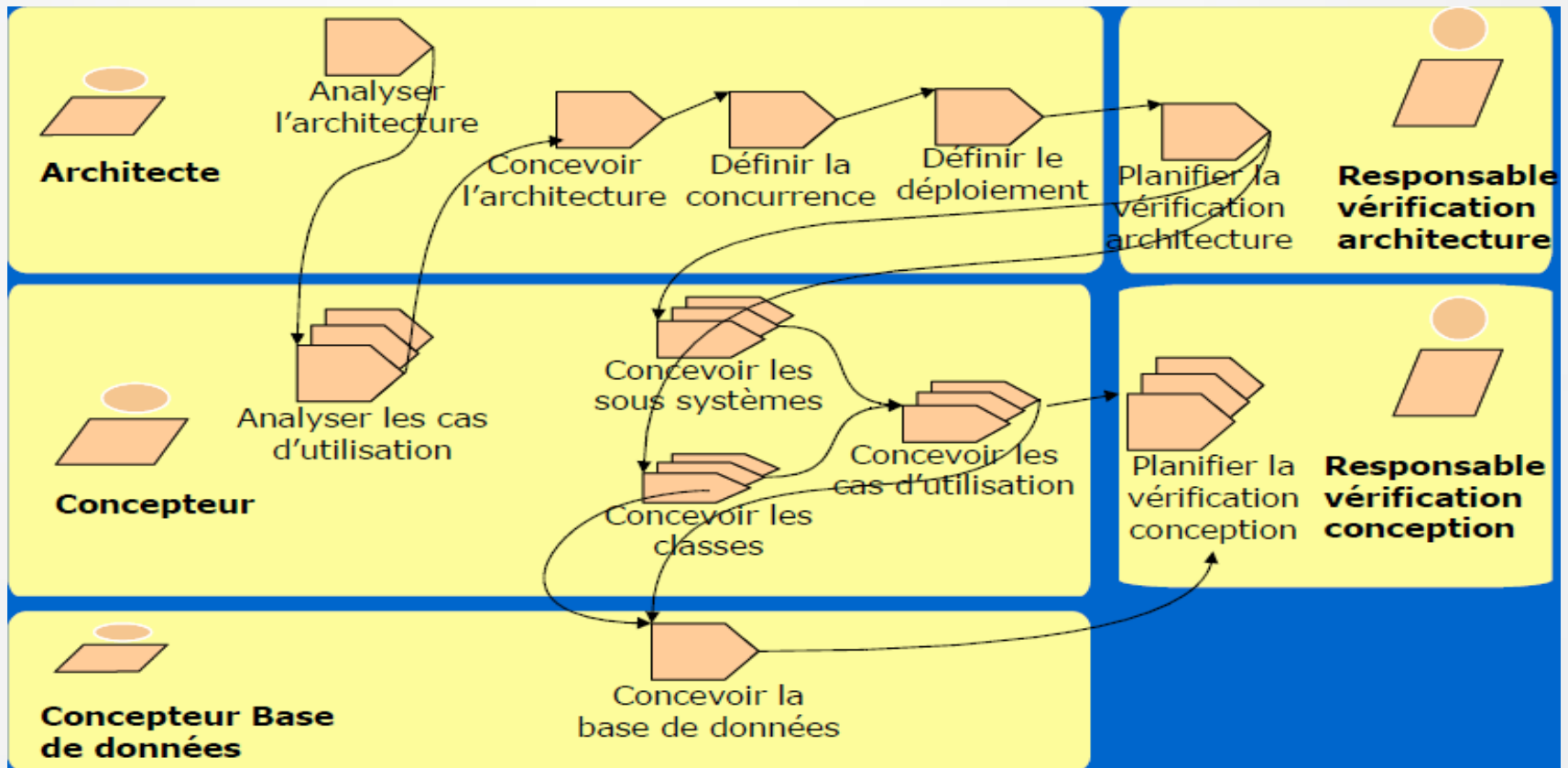
Ressource	Rôle 	Activités 
Paul	Concepteur	Définir Opérations
Marie	Rédacteur. D.Utilisation	Détailler le D. Utilisation
Joseph	Analyste Système	Trouver Acteurs et Cas Util.
Sylvia	Développeur.	Réaliser les tests des unités.
Stefanie	Architecte	Concevoir.

Chaque individu est associé à un ou plusieurs rôles.

► RUP (Rational Unified Process)



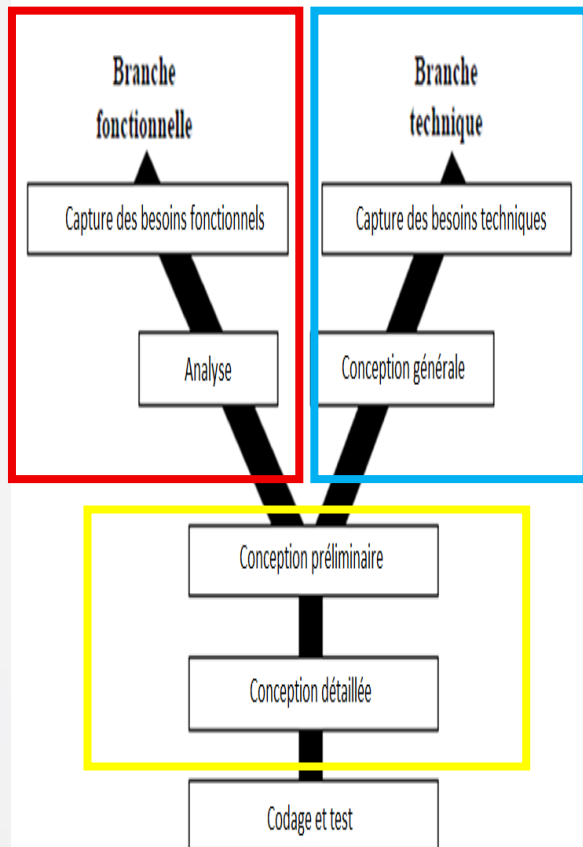
• RUP: Exemple illustratif des activités Analyse et Conception



▶ 2TUP (2 Track Unified Process)



- **Le développement d'un système peut se décomposer suivant :**



- **Une branche fonctionnelle :**

- Capitalise la connaissance du métier de l'entreprise.
- Capture les besoins fonctionnels.

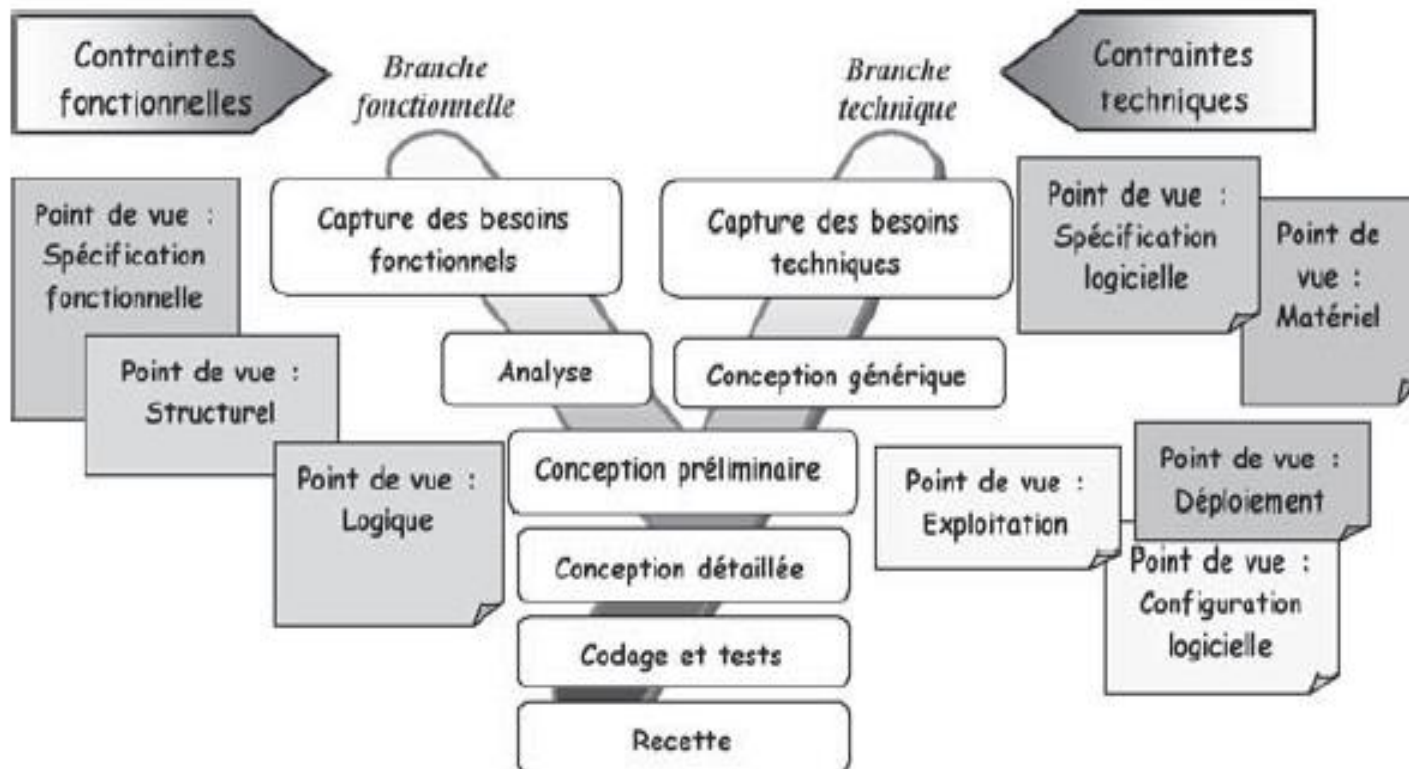
- **Une branche technique :**

- Capitalise un savoir-faire technique et/ou des contraintes techniques.
- Les techniques développées pour le système sont indépendantes des fonctions à réaliser.

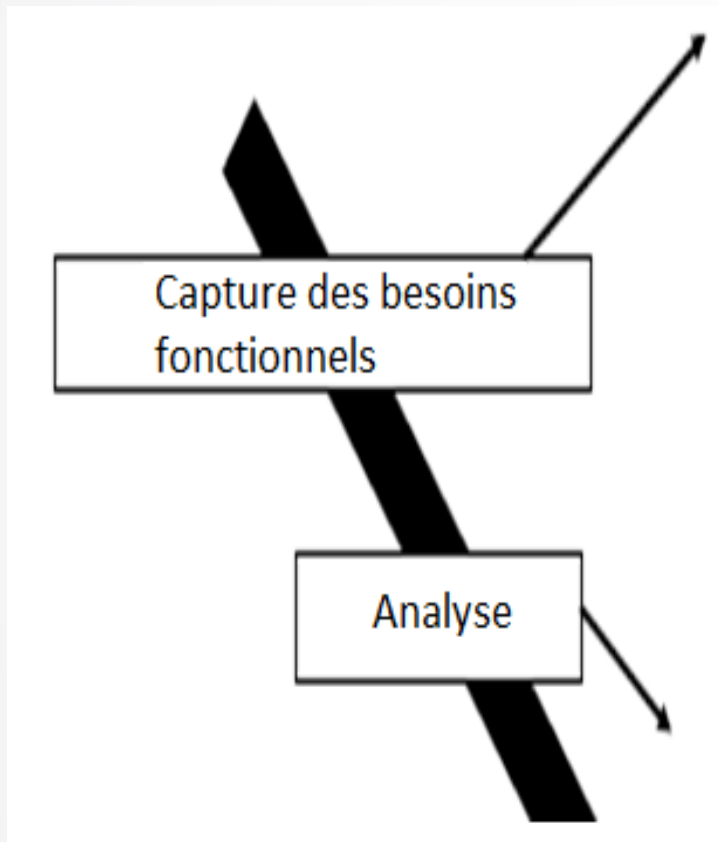
- **La phase de réalisation :**

- Réunir les deux branches, permettant de mener une conception applicative .
- Livrer une solution adaptée aux besoins.

▶ 2TUP (2 Track Unified Process)

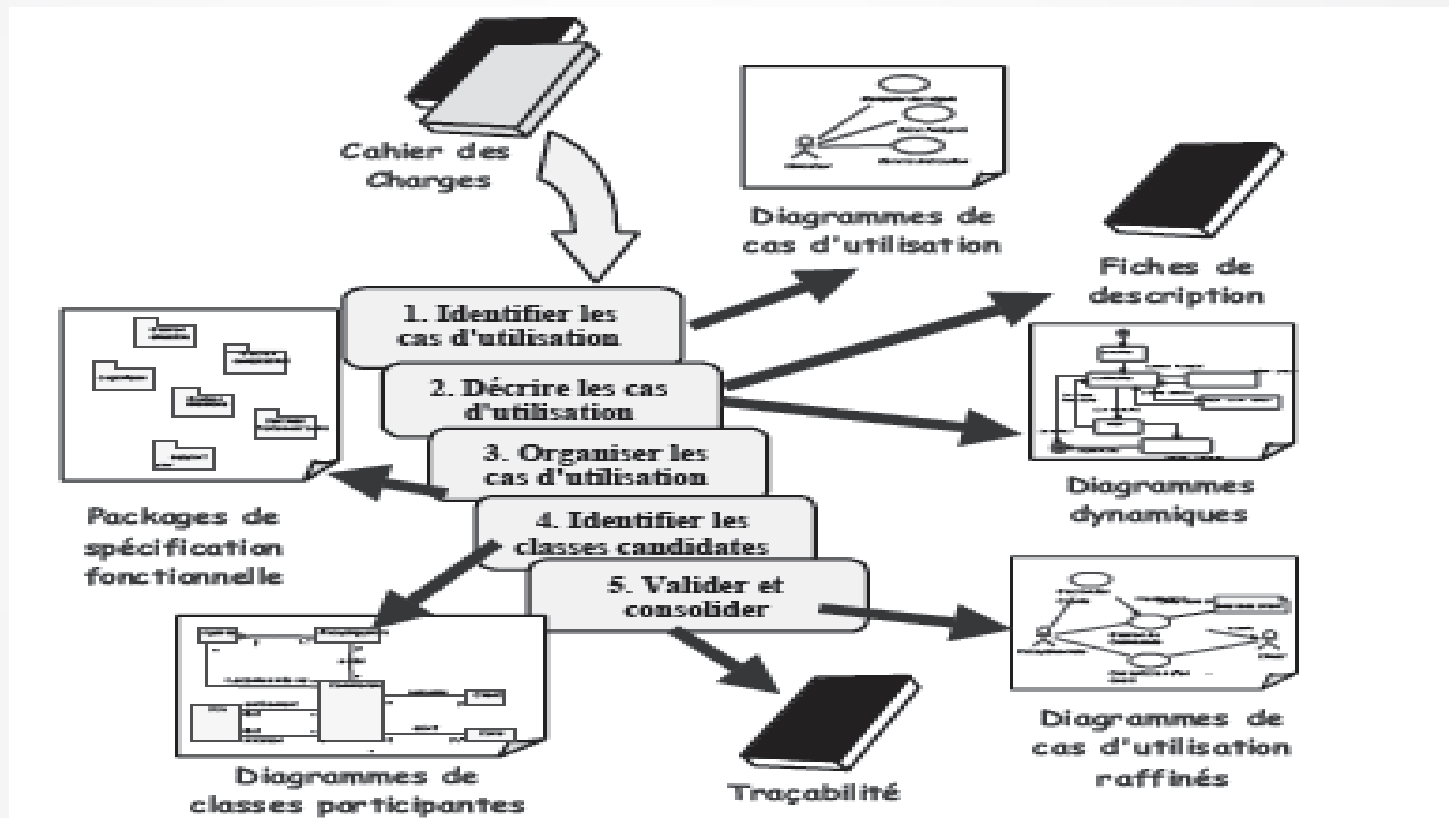


▶ 2TUP (2 Track Unified Process)

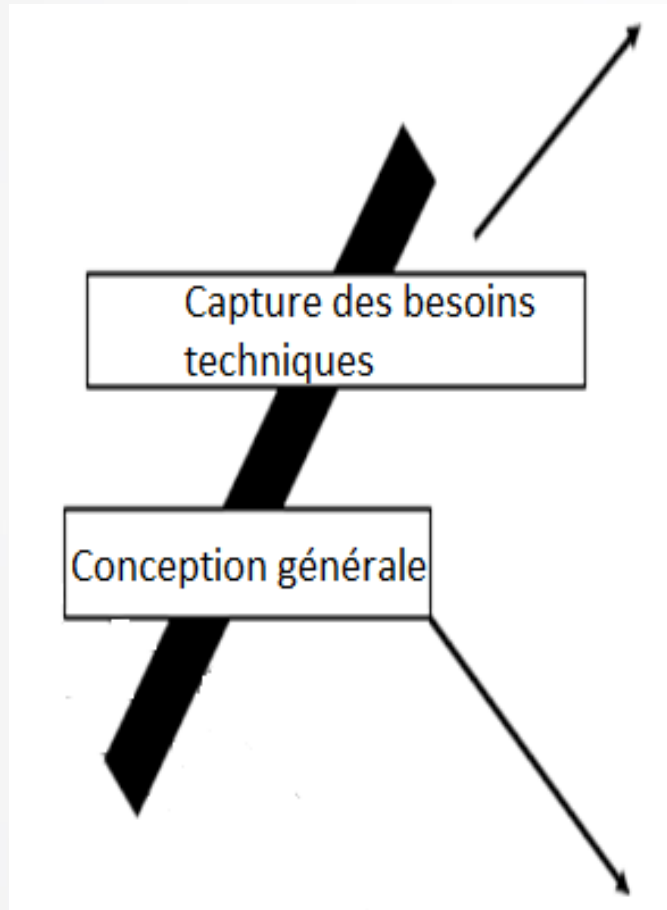


- S'intéresser au métier de l'utilisateur.
- Etudier précisément la spécification fonctionnelle de manière à obtenir une idée de ce que réalise le système sans se soucier des technologies à utiliser.

► 2TUP (2 Track Unified Process)

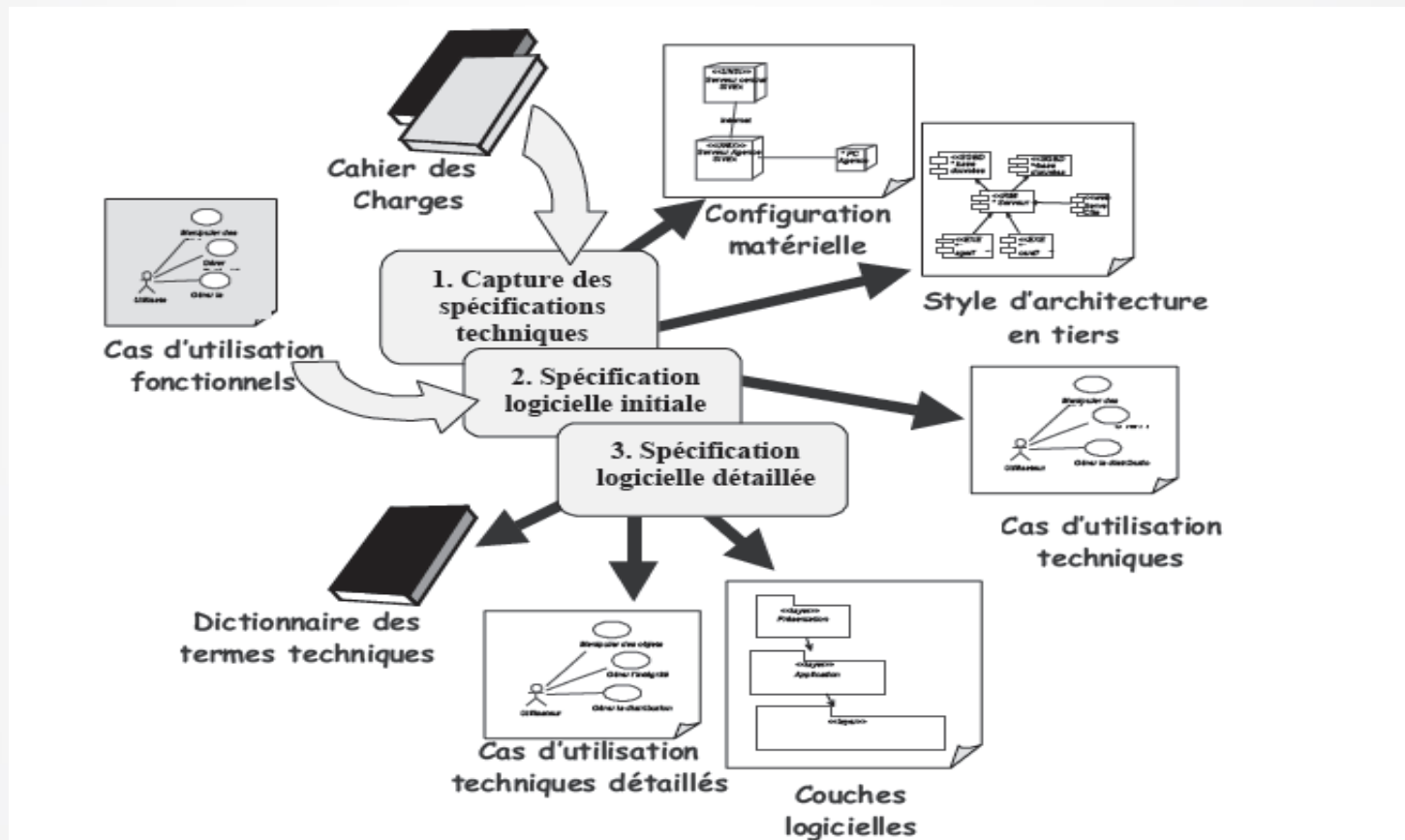


▶ 2TUP (2 Track Unified Process)

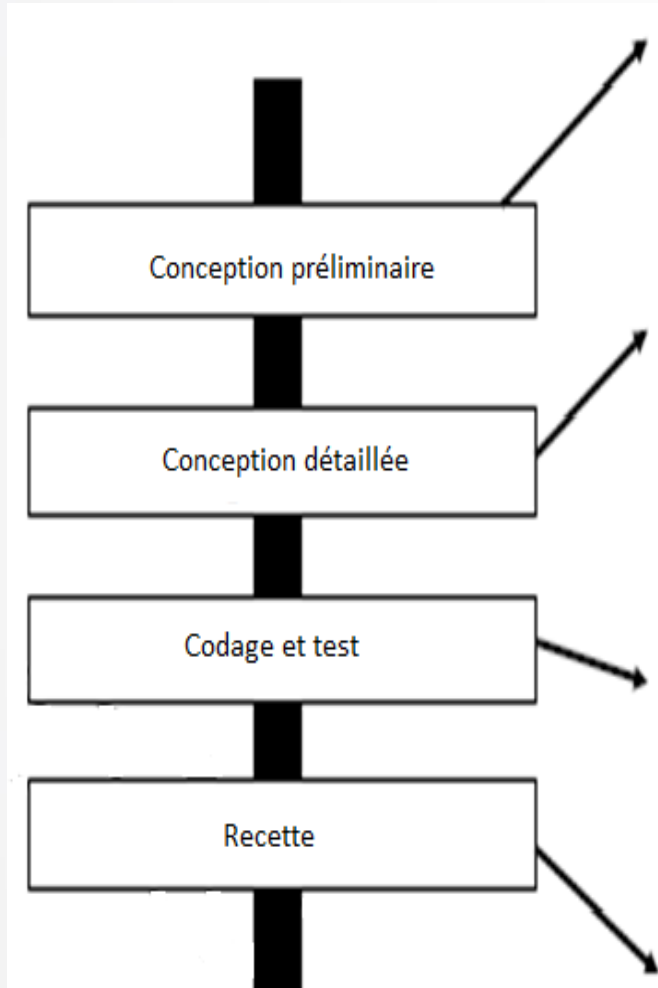


- Voit toutes les contraintes et les choix dimensionnant la conception du système (outils+matériel+contrainte d'intégration).
- Capture des besoins techniques avec l'existant.
- Définit les composants nécessaires à la construction de l'architecture technique.

► 2TUP (2 Track Unified Process)



▶ 2TUP (2 Track Unified Process)



- Intégration des 2 branches.
- Étude de la réalisation de chaque composant.
- Codage des composants + test.
- Validation et vente du produit.



Synthèse : Méthodologie lourde ou méthodologie agile



- **Inconvénients du PU :**

- Fait tout, mais lourd.
- Parfois difficile à mettre en œuvre de façon spécifique.
- UP pour les gros projets qui génèrent beaucoup de documentation.
- Quelles activités pouvons-nous abandonner tout en produisant des logiciels de qualité?
- Comment mieux travailler avec le client pour nous focaliser sur ses besoins les plus prioritaires et être aussi réactifs que possible ?

➡ Processus unifié vs SCRUM

- RUP 4 fois plus lent que Scrum : jusqu'à 27 rôles !
 - beaucoup plus de réunions
 - beaucoup plus de reportings
 - beaucoup plus d'efforts de communication




Méthodes agiles

- Manifeste pour le développement agile de logiciels en 2001 :
 - 4 valeurs.
 - 12 principes.

Les 4 valeurs des méthodes agiles :


- Priorité aux **personnes** et aux **interactions** sur les *procédures et les outils* ;
- Priorité aux **applications fonctionnelles** sur une *documentation pléthorique* ;
- Priorité à la **collaboration avec le client** sur la *négociation de contrat* ;
- Priorité à l'**acceptation du changement** sur la *planification*.



Méthodes agiles

- **Les 12 principes des méthodes agiles :**

1. Priorité à la satisfaction du client à travers la livraison rapide et continue du logiciel.
2. Acceptation des changements même tardifs dans le développement.
3. Livraison fréquente du logiciel, de 2 semaines à 2 mois, avec une préférence pour les périodes courtes.
4. Acteurs métier et développeurs doivent travailler ensemble quotidiennement tout au long du projet.
5. Construire des projets autour de personnes motivés.
6. Le moyen le plus efficace de véhiculer l'information vers et à l'intérieur d'une équipe de développement est la conversation face à face.



Méthodes agiles

7. La métrique principale pour juger de la progression d'un projet est le logiciel fonctionnel.
8. Les processus agiles encouragent le développement durable. Les financeurs, les développeurs et les utilisateurs doivent maintenir un rythme constant indéfiniment.
9. Une attention continue à l'excellence technique et au bon design améliore l'agilité.
10. La simplicité – l'art de maximiser le montant de travail non fait – est essentiel.
11. Les meilleures architectures, besoins et conceptions proviennent des équipes auto-organisées.
12. A intervalles réguliers, l'équipe réfléchit aux manières de devenir plus efficace, puis ajuste ses comportements de façon à s'y conformer.



Exemple de méthode agile : Scrum



- Scrum est une méthode agile utilisée dans le développement de logiciels. Elle vise à **satisfaire au mieux les besoins du client** tout en **maximisant les probabilités de réussite du projet**.
- Un projet utilisant Scrum est composé d'une suite d'itérations courtes de l'ordre de 3 à 6 semaines appelées **sprints**.
- A la fin d'un sprint, l'équipe livre au client un incrément de logiciel fini potentiellement livrable.
- Le projet peut être réorienté par le client à la fin de chaque sprint.
- Le **backlog du produit** constitue l'ensemble du **travail connu sur le projet à un instant t**.
- Le **travail à faire durant un sprint** est listé dans le **backlog du sprint**.

Exemple de méthode agile : Scrum

3 rôles



Product Owner



Scrum Master

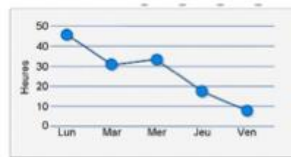


L'équipe

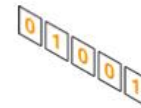
3 artéfacts



backlog



burndown chart

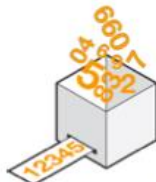


produit

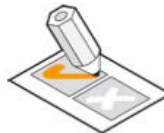


Scrum board

3 réunions



planification de sprint



Daily Scrum



revue de sprint



rétrospective



Exemple de méthode agile : Scrum



Scrum : Les outils

- Les outils dédiés au management de projets agiles exemple IceScrum, Xplanner, etc.
- Le tableau blanc et les post-its :

